Manual Técnico

Entorno de Desarrollo - Backend API

Lenguaje

- **Python 3.x**
  Se recomienda utilizar una versión
  3.10 o superior para asegurar compatibilidad con las dependencias instaladas

Entorno Virtual

- Se utiliza `venv` para aislar el entorno del proyecto:

```
python -m venv .venv
source .venv/bin/activate  # Linux/Mac
.venv\Scripts\activate     # Windows
```

Instalación de Dependencias

- Una vez activado el entorno virtual, instala las dependencias necesarias con:

```
pip install -r requirements.txt
```

Estructura de Proyecto

```
/client
 └── /app
    ├── /users
    │   ├── __init__.py        # Inicialización del módulo users
    │   └── routes.py          # Endpoints HTTP para usuarios
    │
    ├── /products
    │   ├── __init__.py        # Inicialización del módulo products
    │   └── routes.py          # Endpoints HTTP para productos
```

```
    │
    ├── /orders
    │   ├── __init__.py        # Inicialización del módulo orders
    │   ├── routes.py          # Endpoints HTTP para órdenes
    │   ├── services.py        # Lógica de negocio y validaciones para órden
es
    │   └── db_operations.py   # Consultas SQL y manipulación de la base d
e datos de órdenes
    │
    ├── /payments
    │   ├── __init__.py        # Inicialización del módulo payments
    │   └── routes.py          # Endpoints HTTP para pagos
    │
    └── __init__.py            # Función create_app() y registro de blueprints de
los módulos

│
├── utils/              # Funciones auxiliares
│   └── db.py           # Conexión a la base de datos Oracle
    └── .env
│
├── run.py              # Punto de entrada principal para ejecutar la app
└── .venv/              # Entorno virtual
```

Dependencias Python Instaladas (`pip list`)

Paquete	Versión	Descripción breve
Flask	3.1.0	Framework principal para la API.
bcrypt	4.3.0	Hash seguro de contraseñas.
cx_Oracle	8.3.0	Cliente Python para conexión a Oracle Database.
oracledb	3.0.0	Nueva versión cliente para Oracle, alternativa a cx_Oracle.
Werkzeug	3.1.3	Proporciona herramientas WSGI para la ejecución de la app.
Otros	Blinker, Click, Jinja2, MarkupSafe, Cryptography, etc.	

Configuración de la Conexión a la Base de Datos con `.env`

Descripción General

El proyecto utiliza un archivo `.env` para gestionar de manera segura y flexible las **credenciales de la base de datos Oracle** y otros parámetros de conexión. Esto evita incluir credenciales sensibles directamente en el código fuente, mejorando la **seguridad**, **mantenibilidad** y facilitando la **configuración de distintos entornos** (desarrollo, producción, etc.).

Contenido del Archivo `.env`

Ubicado en `/client/utils/.env`:

```
DB_USER=SYSTEM
DB_PASSWORD=1234E
DB_HOST=localhost
DB_PORT=1521
DB_SERVICE=XE
```

Configuración de `utils/db.py`

El archivo `db.py` es el responsable de establecer la **conexión a Oracle Database**, cargando previamente las variables desde el archivo `.env`.

Código Ejemplo:

```
import cx_Oracle
from dotenv import load_dotenv
import os

env_path = os.path.join(os.path.dirname(__file__), '.env')
load_dotenv(dotenv_path=env_path)

def get_connection():
    db_user = os.getenv('DB_USER')
    db_password = os.getenv('DB_PASSWORD')
    db_host = os.getenv('DB_HOST')
    db_port = os.getenv('DB_PORT')
```

```
    db_service = os.getenv('DB_SERVICE')

    dsn_tns = cx_Oracle.makedsn(db_host, db_port, service_name=db_servi
ce)

    conn = cx_Oracle.connect(user=db_user, password=db_password, dsn=
dsn_tns)
    return conn
```

▼ Endpoints REST

▼ Gestión de usuarios

estos endpoints gestionan el ciclo de vida de los usuarios en la plataforma, desde el registro, autenticación, consulta y actualización, hasta el manejo de direcciones asociadas.

URL Base del Módulo:

```
/api/users
```

Endpoints de Usuarios

#	Método	Endpoint	Descripción
1.1	POST	`/api/users`	Crear usuario
1.2	POST	`/api/users/login`	Autenticar usuario (login)
1.3	GET	`/api/users/<int:id>`	Obtener perfil de usuario
1.4	PUT	`/api/users/<int:id>`	Actualizar usuario
1.5	DELETE	`/api/users/<int:id>`	Eliminar/inactivar usuario
1.6	POST	`/api/users/add_address`	Agregar dirección a un usuario

1.1 Crear Usuario (Registro)

Método	Endpoint
`POST`	`/api/users`

Crea un nuevo usuario en la plataforma, junto a su correo electrónico asociado.

Request JSON

```
{
  "username": "jdoe",
  "email": "jdoe@example.com",
  "password": "secret123",
  "phone": "12345678",
  "national_document": "A12345678",
  "name": "John",
  "lastname": "Doe"
}
```

Proceso Interno

- Verifica que el `username` y `email` no existan en la base de datos.
- Hashea la contraseña con `bcrypt`.
- Crea el usuario en la tabla `CLIENTE`.
- Crea el correo en la tabla `EMAILCLIENTE`.
- Marca el campo `active` en `CLIENTE` y `confirmed` en `EMAILCLIENTE` como `1`.

Respuesta Exitosa

```
{
  "status": "success",
  "message": "User created successfully",
  "user_id": 1
}
```

Códigos de Respuesta

Código	Descripción
201	Usuario creado exitosamente.
400	Datos incompletos o inválidos.
409	El username o email ya existe.
500	Error interno al crear el usuario.

1.2 Iniciar Sesión (Login)

Método	Endpoint
`POST`	`/api/users/login`

Autentica al usuario mediante `username` y `password`.

Request JSON

```
{
  "username": "jdoe",
  "password": "secret123"
}
```

Proceso Interno

- Busca el usuario por `username`.
- Compara el hash de la contraseña usando `bcrypt`.
- Devuelve un `sessionId` simulado (por ahora).

Respuesta Exitosa

```
json
CopiarEditar
{
  "status": "success",
  "message": "User authenticated",
  "sessionId": "dummy_session_1"
}
```

Códigos de Respuesta

Código	Descripción
200	Login exitoso.
400	Campos faltantes (`username` o `password`).
401	Credenciales inválidas.

1.3 Obtener Perfil de Usuario

Método	Endpoint
`GET`	`/api/users/<int:id>`

Obtiene la información básica de un usuario específico, sin exponer la contraseña.

Request

- `:id`: ID del usuario (parámetro en la URL).

Respuesta Exitosa

```
{
  "id": 10,
  "username": "jdoe",
  "phone": "12345678",
  "createdAt": "2025-02-01T10:00:00Z"
}
```

Códigos de Respuesta

Código	Descripción
200	Perfil retornado exitosamente.
404	Usuario no encontrado.

1.4 Actualizar Usuario

Método	Endpoint
`PUT`	`/api/users/<int:id>`

Método	Endpoint

Actualiza el `phone` y/o el `email` del usuario.

Request JSON

```
{
  "phone": "87654321",
  "email": "john@example.com"
}
```

Proceso Interno

- Verifica la existencia del usuario.
- Actualiza solo los campos enviados (`phone` y/o `email`).

Respuesta Exitosa

```
{
  "status": "success",
  "message": "User updated successfully"
}
```

Códigos de Respuesta

Código	Descripción
200	Usuario actualizado.
400	Nada que actualizar.
404	Usuario no encontrado.

1.5 Eliminar/Inactivar Usuario

Método	Endpoint
`DELETE`	`/api/users/<int:id>`

Marca como inactivo (`active = 0`) el usuario indicado.

Respuesta Exitosa

```
{
  "status": "success",
  "message": "User deleted/inactivated successfully"
}
```

Códigos de Respuesta

Código	Descripción
200	Usuario inactivado.
404	Usuario no encontrado.

1.6 Agregar Dirección a un Usuario

Método	Endpoint
`POST`	`/api/users/add_address`

Registra una nueva dirección para el usuario, buscándolo por `username`.

Request JSON

```
{
  "username": "jdoe",
  "address": "123 Main Street, City, Country"
}
```

Proceso Interno

- Verifica la existencia del usuario (`username`) y que esté activo.
- Inserta la dirección en la tabla `DIRECCION` relacionada al `client_id`.

Respuesta Exitosa

```
{
  "status": "success",
  "message": "Address added for user 'jdoe'"
}
```

Códigos de Respuesta

Código	Descripción
201	Dirección creada exitosamente.
400	Faltan datos en el request.
404	Usuario no encontrado.

▼ Gestión de Productos

estos endpoints gestionan el ciclo de vida de los productos en la plataforma, desde la información general, por id, creación, actualizacion, eliminación.

URL Base del Módulo:

```
/api/products
```

Endpoints de Productos

Método	Endpoint	descripcion
`GET`	`/api/products`	Listar productos con inventario
`GET`	`/api/products/:id`	Detalle de un producto
`POST`	`/api/products`	Crear un nuevo producto
`PUT`	`/api/products/:id`	Actualizar precio y/o stock
`DELETE`	`/api/products/:id`	Eliminar (inactivar) un producto

2.1 Listar Productos (GET)

- **Endpoint**: `/api/products`
- **Método**: `GET`
- **Descripción**: Retorna la lista completa de productos activos, junto con su categoría e inventario distribuido por sede.

Request

No requiere parámetros.

Response (JSON)

```
{
  "products": [
    {
      "id": 1,
      "name": "Laptop X",
      "price": 750.00,
      "category": "Electrónicos",
      "inventory": [
        {
          "location": "Sede Central",
          "stock": 5
        },
        {
          "location": "Sede Norte",
          "stock": 3
        }
      ]
    },
    {
      "id": 2,
      "name": "Smartphone Y",
      "price": 500.00,
      "category": "Electrónicos",
      "inventory": [
        {
          "location": "Sede Central",
          "stock": 10
        },
        {
          "location": "Sede Sur",
          "stock": 7
        }
      ]
    }
  ]
}
```

Códigos de Respuesta

Código	Significado
200	Productos retornados con éxito
500	Error interno en el servidor

2.2 Detalle de Producto (GET)

- **Endpoint**: `/api/products/:id`
- **Método**: `GET`
- **Descripción**: Obtiene la información detallada de un producto, incluyendo su inventario en cada sede.

Request

- Parámetro en URL: `:id` → ID del producto.

Response (JSON)

```
json
CopiarEditar
{
  "id": 1,
  "name": "Laptop X",
  "description": "Laptop de alto rendimiento",
  "price": 750.00,
  "category": "Electrónicos",
  "inventory": [
    {
      "location": "Sede Central",
      "stock": 5
    },
    {
      "location": "Sede Norte",
      "stock": 3
    }
  ]
```

```
}
```

Códigos de Respuesta

Código	Significado
200	Producto encontrado
404	Producto no encontrado
500	Error interno en el servidor

2.3 Crear Producto (POST)

- **Endpoint**: `/api/products`
- **Método**: `POST`
- **Descripción**: Crea un nuevo producto en la base de datos y asigna su inventario inicial en una sede específica.

Request (JSON)

```
{
  "name": "Monitor Gamer 27\"",
  "description": "Monitor curvo de 27 pulgadas, 165Hz, 1ms de respu
esta",
  "price": 350.00,
  "stock": 15,
  "category": "Electrónicos",
  "location": "Sede Central"
}
```

Proceso Interno

1. Verifica que la **categoría** exista en la tabla `CATEGORIA`.
2. Verifica que la **sede** exista en la tabla `SEDE`.
3. Inserta el producto en la tabla `PRODUCTO`.
4. Inserta el inventario inicial en la tabla `INVENTARIO`.

Response (JSON)

```
{
  "status": "success",
  "message": "Product created successfully",
  "productId": 5
}
```

Códigos de Respuesta

Código	Significado
201	Producto creado exitosamente
400	Faltan campos requeridos
404	Categoría o sede no encontrada
500	Error interno en el servidor

2.4 Actualizar Producto (PUT)

- **Endpoint**: `/api/products/:id`
- **Método**: `PUT`
- **Descripción**: Modifica el precio del producto y/o su stock en una sede específica.

Request (JSON)

```
json
CopiarEditar
{
  "price": 700.00,
  "stock": 20,
  "location": "Sede Central"
}
```

- Puedes enviar solo el `price`, o solo el `stock` con `location`, o ambos.

Proceso Interno

1. Actualiza el `price` en la tabla `PRODUCTO` si se envía.
2. Si se envía `stock` y `location`:
    - Verifica si el inventario ya existe.
    - Si existe, actualiza el `quantity` en la tabla `INVENTARIO`.
    - Si no existe, crea un nuevo registro en `INVENTARIO`.

Response (JSON)

```
{
  "status": "success",
  "message": "Product and/or inventory updated successfully"
}
```

Códigos de Respuesta

Código	Significado
200	Producto o inventario actualizado con éxito
400	Faltan campos necesarios (`price` o `stock` / `location`)
404	Producto o sede no encontrada
500	Error interno en el servidor

2.5 Eliminar Producto (DELETE)

- **Endpoint**: `/api/products/:id`
- **Método**: `DELETE`
- **Descripción**: Marca como inactivo un producto para que no aparezca en el listado, sin eliminarlo físicamente.

Request

- Parámetro en URL: `:id` → ID del producto a inactivar.

Proceso Interno

1. Cambia el campo `active` a `0` en la tabla `PRODUCTO`.

Response (JSON)

```
{
  "status": "success",
  "message": "Product deleted/inactivated successfully"
}
```

Códigos de Respuesta

Código	Significado
200	Producto inactivado con éxito
404	Producto no encontrado
500	Error interno en el servidor

Requisitos previos

1. Las **categorías** deben estar creadas previamente en `CATEGORIA`.
2. Las **sedes** deben existir en `SEDE`.

▼ Gestión de Órdenes de Compra

Estos endpoints gestionan el ciclo de vida de las órdenes en la plataforma: desde la creación de una orden de compra hasta la consulta de todas las órdenes y el detalle de una específica.

URL Base del Módulo:

```
/api/orders
```

Endpoints de Órdenes

#	Método	Endpoint	Descripción
3.1	POST	`/api/orders`	Crear una nueva orden de compra
3.2	GET	`/api/orders`	Listar todas las órdenes de compra
3.3	GET	`/api/orders/<int:id>`	Detalle de una orden específica

3.1 Crear Orden de Compra

Método	Endpoint
`POST`	`/api/orders`

Crea una nueva orden de compra para un usuario. Se validan los datos del usuario, la existencia del método de pago y el stock de los productos solicitados.

Request JSON

```
{
  "userId": 4,
  "items": [
    { "productId": 5, "quantity": 2 },
    { "productId": 6, "quantity": 1 }
  ],
  "shippingAddress": "Calle Ficticia 123, Ciudad",
  "paymentMethodId": 1
}
```

Proceso Interno

- Verifica que el `userId` exista y esté activo.
- Valida que el `paymentMethodId` exista y esté vinculado al usuario.
- Se elige automáticamente la sede con **mayor stock disponible** para el primer producto de la orden (asignando `location_id`).
- Crea la orden en `ORDENCOMPRA`.
- Por cada ítem:
  - Valida el stock disponible en la sede.
  - Inserta el detalle de la orden en `DETALLEORDEN`.
  - Actualiza el inventario (`INVENTARIO`) restando el stock.
- Calcula el `totalAmount` de la orden.

Respuesta Exitosa

```
{
  "status": "success",
  "message": "Order created successfully",
  "orderId": 101,
  "totalAmount": 1800.00,
  "orderStatus": "processing"
}
```


Códigos de Respuesta

Código	Descripción
201	Orden creada exitosamente.
400	Faltan campos obligatorios o datos inválidos.
404	Usuario, método de pago o producto inexistente.
500	Error interno al crear la orden.

3.2 Listar Órdenes

Método	Endpoint
`GET`	`/api/orders`

Retorna un listado con todas las órdenes creadas en el sistema. Cada orden incluye su `orderId`, `userId`, `totalAmount` y `createdAt`.

Proceso Interno

- Consulta la tabla `ORDENCOMPRA` y suma los subtotales de `DETALLEORDEN` para obtener el `totalAmount`.
- Ordena los resultados por fecha de creación (`created_at DESC`).

Respuesta Exitosa


```
{
  "orders": [
    {
      "orderId": 101,
      "userId": 4,
```

```
    "totalAmount": 1800.00,
    "createdAt": "2025-03-12"
   },
   {
    "orderId": 102,
    "userId": 5,
    "totalAmount": 500.00,
    "createdAt": "2025-03-13"
   }
  ]
}
```

Códigos de Respuesta

Código	Descripción
200	Listado de órdenes retornado.
500	Error interno al consultar órdenes.

3.3 Detalle de Orden

Método	Endpoint
`GET`	`/api/orders/<int:id>`

Retorna el detalle completo de una orden específica, incluyendo los productos, cantidades y precios de cada ítem.

Proceso Interno

- Valida que el `order_id` exista en `ORDENCOMPRA`.
- Consulta los ítems asociados en `DETALLEORDEN` y sus precios.
- Calcula el `totalAmount` de la orden.

Respuesta Exitosa

```
{
  "orderId": 101,
  "userId": 4,
  "items": [
```

```
    { "productId": 5, "quantity": 2, "price": 750.00 },
    { "productId": 6, "quantity": 1, "price": 500.00 }
  ],
  "totalAmount": 2000.00,
  "createdAt": "2025-03-12"
}
```

Códigos de Respuesta

Código	Descripción
200	Detalle de orden retornado.
404	Orden no encontrada.
500	Error interno al consultar detalle.

▼ Gestion de Pagos

Estos endpoints gestionan el ciclo de vida de los pagos en la plataforma: desde el registro de un pago para una orden de compra hasta la consulta de pagos con filtros opcionales.

URL Base del Módulo:

```
/api/payments
```

Endpoints de Pagos

#	Método	Endpoint	Descripción
1.1	POST	`/api/payments`	Registrar un pago para una orden de compra
1.2	GET	`/api/payments`	Listar pagos registrados, con posibilidad de aplicar filtros por orden, método o fechas

1.1 Registrar Pago

Método	Endpoint
`POST`	`/api/payments`

Registra un pago para una orden de compra. Se validan la existencia de la orden y del método de pago, se verifica que el pago no exceda el monto pendiente, y en caso de completar el pago de la orden se actualiza (o inserta) la información de envío asociada.

Request JSON

```
{
  "orderId": 101,
  "amount": 500.00,
  "paymentMethodId": 2
}
```

Proceso Interno

- **Validación de campos:**

  Se verifica que se incluyan `orderId`, `amount` y `paymentMethodId`.

- **Verificación de existencia:**
  - Se comprueba que la orden exista en la tabla `OrdenCompra`.
  - Se valida que el método de pago exista en la tabla `MetodoPago`.
- **Cálculo del monto pendiente:**
  - Se obtiene el monto total de la orden (suma de los subtotales de cada ítem).
  - Se calcula la suma de los pagos ya registrados.
  - Se determina el monto pendiente y se verifica que el pago a registrar no lo exceda.
- **Determinación del estado del pago:**

  Se calcula el nuevo total pagado.
  - Si el total pagado (previo + pago actual) es mayor o igual al monto total de la orden, se establece el estado como **aprobado** (estado_pago_id = 1).

    - En caso contrario, el estado se marca como **pendiente** (estado_pago_id = 0).

- **Registro del pago:**

  Se inserta el nuevo registro en la tabla `Pago`.

- **Gestión de envío:**

  Si el pago completa el monto de la orden, se:

    - Actualiza el registro de envío existente (si existe) o
    - Inserta un nuevo registro en la tabla `Envio` con datos generados (como número de guía, fecha de despacho, etc.).

- **Commit de la transacción:**

  Se confirma la operación en la base de datos.

Respuesta Exitosa

```
{
  "status": "success",
  "message": "Payment registered successfully",
  "paymentId": 55,
  "paymentStatus": "approved",
  "totalPaid": 1500.00,
  "remainingAmount": 0.00
}
```

Códigos de Respuesta

Código	Descripción
201	Pago registrado exitosamente.
400	Campos faltantes o pago excede el monto pendiente.
404	Orden o método de pago no encontrado.
500	Error interno al registrar el pago.

1.2 Listar Pagos

Método	Endpoint
`GET`	`/api/payments`

Retorna un listado de pagos registrados en el sistema, permitiendo filtrar por orden, método de pago y rango de fechas.

Parámetros de Consulta (Query Parameters)

- **orderId** (opcional, entero):
  Filtra los pagos asociados a una orden específica.
- **paymentMethodId** (opcional, entero):
  Filtra por el método de pago utilizado.
- **dateFrom** (opcional, string - formato `YYYY-MM-DD`):
  Retorna pagos registrados a partir de esta fecha.
- **dateTo** (opcional, string - formato `YYYY-MM-DD`):
  Retorna pagos registrados hasta esta fecha.

Proceso Interno

- Se consulta la tabla `Pago` y se aplican los filtros recibidos.
- Se realiza un JOIN con las tablas `MetodoPago` y `EstadoPago` para obtener el nombre del método y el estado del pago.
- Se ordenan los resultados por fecha de creación en orden descendente.

Respuesta Exitosa

```
json
Copiar
{
  "payments": [
    {
      "paymentId": 55,
      "orderId": 101,
      "amount": 500.00,
      "method": "Tarjeta de Crédito",
      "status": "approved",
      "createdAt": "2025-03-12T15:30:00Z"
```

```
    },
    {
      "paymentId": 56,
      "orderId": 102,
      "amount": 250.00,
      "method": "Transferencia",
      "status": "pending",
      "createdAt": "2025-03-13T10:15:00Z"
    }
  ]
}
```

Códigos de Respuesta

Código	Descripción
200	Listado de pagos retornado exitosamente.
500	Error interno al consultar pagos.

Esta documentación provee una guía clara para el uso de los endpoints del módulo de pagos, facilitando la integración y el mantenimiento de la funcionalidad relacionada con el registro y la consulta de pagos en la plataforma.

▼ Scripts SQL Utilizados

1. Módulo de Pagos (`db_payments.py`)

- **get_order_by_id(cursor, order_id)**
  - **Consulta:**

    ```
    SELECT id
    FROM OrdenCompra
    WHERE id = :order_id
    ```

  - **Propósito:** Recuperar el identificador de una orden de compra dado su ID.
- **get_payment_method_by_id(cursor, payment_method_id)**

- **Consulta:**

```
SELECT id
FROM MetodoPago
WHERE id = :payment_method_id
```

- **Propósito:** Verificar la existencia de un método de pago específico.

- **get_total_order_amount(cursor, order_id)**
  - **Consulta:**

```
SELECT NVL(SUM(subtotal), 0)
FROM DetalleOrden
WHERE order_id = :order_id
```

  - **Propósito:** Calcular el monto total de una orden sumando los subtotales de cada detalle. Se utiliza `NVL` para asegurar que se devuelva 0 en ausencia de registros.

- **get_total_paid_amount(cursor, order_id)**
  - **Consulta:**

```
SELECT NVL(SUM(total_amount), 0)
FROM Pago
WHERE order_id = :order_id
```

  - **Propósito:** Determinar el total pagado hasta el momento para una orden dada.

- **insert_payment(cursor, order_id, payment_method_id, estado_pago_id, amount)**
  - **Consulta:**

```
INSERT INTO Pago (
    id, order_id, metodo_pago_id, estado_pago_id,
    total_amount, created_at, updated_at
)
VALUES (
    PAGO_SEQ.NEXTVAL, :order_id, :payment_method_id, :estado
```

```
_pago_id,
    :total_amount, :created_at, :created_at
)
```

- **Propósito:** Registrar un nuevo pago para una orden utilizando la secuencia `PAGO_SEQ` para generar un nuevo identificador.

- **get_envio_by_order_id(cursor, order_id)**
  - **Consulta:**

```
SELECT id
FROM Envio
WHERE order_id = :order_id
```

  - **Propósito:** Obtener el registro de envío asociado a una orden.

- **update_envio(cursor, envio_id, order_id)**
  - **Consulta:**

```
UPDATE Envio
SET company_id = :company_id,
    shipping_status_id = :shipping_status_id,
    number_company_guide = :guide_number,
    dispatch_date = :dispatch_date,
    delivered_at = :delivered_at,
    updated_at = :updated_at
WHERE id = :envio_id
```

  - **Propósito:** Actualizar los datos del envío (como número de guía y fechas) para una orden determinada.

- **insert_envio(cursor, order_id)**
  - **Consulta:**

```
INSERT INTO Envio (
    id, order_id, company_id, shipping_status_id, address,
    number_company_guide, dispatch_date, delivered_at,
    created_at, updated_at
)
```

```
VALUES (
    ENVIO_SEQ.NEXTVAL, :order_id, :company_id, :shipping_statu
s_id,
    'Dirección pendiente', :guide_number, :dispatch_date, :deliver
ed_at,
    :created_at, :created_at
)
```

- **Propósito:** Insertar un nuevo registro de envío asociado a una orden, asignando valores predeterminados para la dirección y el número de guía.

- **list_payments(cursor, filters)**
  - **Consulta:**

```
SELECT
    p.id AS payment_id,
    p.order_id,
    p.total_amount,
    mp.name AS payment_method,
    ep.name AS payment_status,
    p.created_at
FROM Pago p
JOIN MetodoPago mp ON mp.id = p.metodo_pago_id
JOIN EstadoPago ep ON ep.id = p.estado_pago_id
WHERE 1 = 1
-- Filtros opcionales (orderId, paymentMethodId, dateFrom, date
To) se agregan dinámicamente
ORDER BY p.created_at DES
```

  - **Propósito:** Listar los pagos registrados aplicando filtros opcionales por orden, método de pago y rango de fechas.

---

2. Módulo de Productos (`db_products.py`)

- **get_all_active_products(cursor)**
  - **Consulta:**

```
SELECT p.id, p.name, p.price, c.name AS category
FROM PRODUCTO p
JOIN CATEGORIA c ON p.category_id = c.id
WHERE p.active = 1
```

  - **Propósito:** Recuperar la lista de productos activos junto a su precio y la categoría asociada.

- **get_inventory_by_product(cursor, product_id)**
  - **Consulta:**

```
SELECT s.name, i.quantity
FROM INVENTARIO i
JOIN SEDE s ON i.location_id = s.id
WHERE i.product_id = :product
```

  - **Propósito:** Obtener la cantidad disponible y la ubicación (sede) para un producto específico.

- **get_product_details(cursor, product_id)**
  - **Consulta:**

```
SELECT p.id, p.name, p.description, p.price,
   NVL(SUM(i.quantity), 0) AS stock,
   c.name AS category
FROM PRODUCTO p
LEFT JOIN INVENTARIO i ON p.id = i.product_id
JOIN CATEGORIA c ON p.category_id = c.id
WHERE p.id = :id AND p.active = 1
GROUP BY p.id, p.name, p.description, p.price, c.nam
```

  - **Propósito:** Obtener detalles completos de un producto, incluyendo descripción, precio, stock total y categoría.

- **get_category_id(cursor, category_name)**
  - **Consulta:**

```
SELECT id FROM CATEGORIA WHERE name = :category
```

- **Propósito:** Determinar el identificador de una categoría según su nombre.

- **get_location_id(cursor, location_name)**
  - **Consulta:**

```
SELECT id FROM SEDE WHERE name = :location
```

  - **Propósito:** Recuperar el identificador de una sede según su nombre.

- **insert_product(cursor, sku, name, description, price, slug, category_id, created_at)**
  - **Consulta:**

```
INSERT INTO PRODUCTO (
    id, sku, name, description, price, slug, active, category_id, cre
ated_at, updated_at
)
VALUES (
    PRODUCTO_SEQ.NEXTVAL, :sku, :name, :description, :price, :
slug, 1, :category_id, :created_at, :created_at
)
```

  - **Propósito:** Registrar un nuevo producto en la base de datos.

- **insert_inventory(cursor, product_id, location_id, quantity, created_at)**
  - **Consulta:**

```
INSERT INTO INVENTARIO (id, quantity, product_id, location_id,
created_at, updated_at)
VALUES (INVENTARIO_SEQ.NEXTVAL, :quantity, :product_id, :loc
ation_id, :created_at, :created_at
```

  - **Propósito:** Registrar una entrada en el inventario para un producto en una sede determinada.

- **update_product_price(cursor, product_id, price, updated_at)**

- **Consulta:**

```
UPDATE PRODUCTO SET price = :price, updated_at = :updated_
at WHERE id = :i
```

- **Propósito:** Actualizar el precio de un producto.

- **get_inventory_entry(cursor, product_id, location_id)**
  - **Consulta:**

```
SELECT id FROM INVENTARIO
WHERE product_id = :product_id AND location_id = :location_id
```

  - **Propósito:** Verificar si ya existe una entrada de inventario para un producto en una sede.
- **update_inventory(cursor, product_id, location_id, quantity, updated_at)**
  - **Consulta:**

```
UPDATE INVENTARIO
SET quantity = :quantity, updated_at = :updated_at
WHERE product_id = :product_id AND location_id = :location_id
```

  - **Propósito:** Actualizar la cantidad de producto disponible en una entrada de inventario.
- **insert_inventory_entry(cursor, product_id, location_id, quantity, created_at)**
  - **Consulta:**

```
INSERT INTO INVENTARIO (id, quantity, product_id, location_id,
created_at, updated_at)
VALUES (INVENTARIO_SEQ.NEXTVAL, :quantity, :product_id, :loc
ation_id, :created_at, :created_at)
```

  - **Propósito:** Insertar una nueva entrada de inventario.
- **deactivate_product(cursor, product_id, updated_at)**

- **Consulta:**

```
UPDATE PRODUCTO SET active = 0, updated_at = :updated_at
WHERE id = :id
```

- **Propósito:** Marcar un producto como inactivo en la base de datos.

---

3. Módulo de Órdenes de Compra (`db_operations.py`)

- **validate_user(cursor, user_id)**
  - **Consulta:**

```
SELECT id FROM CLIENTE WHERE id = :id AND active = 1
```

  - **Propósito:** Confirmar que el usuario existe y se encuentra activo.
- **validate_payment_method(cursor, payment_method_id, user_id)**
  - **Consulta:**

```
SELECT mp.id FROM MetodoPago mp
JOIN MetodoPagoCliente mpc ON mpc.id = mp.payment_client_i
d
WHERE mp.id = :payment_method_id AND mpc.client_id = :user_
```

  - **Propósito:** Verificar que el método de pago existe y está asociado al usuario.
- **create_order_record(cursor, user_id, location_id)**
  - **Consulta:**

```
INSERT INTO OrdenCompra (id, client_id, location_id, created_a
t, updated_at)
VALUES (ORDENCOMPRA_SEQ.NEXTVAL, :client_id, :location_id,
SYSDATE, SYSDATE)
```

  - **Propósito:** Crear un registro en la tabla de órdenes de compra y obtener el nuevo identificador usando la secuencia `ORDENCOMPRA_SEQ`.

- **process_order_items(cursor, order_id, items)**
  - **Consultas incluidas:**
    - **Obtener precio del producto:**

```
SELECT price FROM PRODUCTO
WHERE id = :product_id AND active = 1
```

    - **Insertar detalle de la orden:**

```
INSERT INTO DetalleOrden (
    id, order_id, product_id, quantity, unit_price, subtotal, crea
ted_at, updated_at
)
VALUES (
    DETALLEORDEN_SEQ.NEXTVAL, :order_id, :product_id, :q
uantity, :unit_price, :subtotal, SYSDATE, SYSDATE
)
```

  - **Propósito:** Procesar cada ítem de la orden, calculando subtotales y acumulando el total de la orden.
- **create_payment(cursor, order_id, payment_method_id, total_amount)**
  - **Consulta:**

```
INSERT INTO Pago (id, order_id, metodo_pago_id, estado_pago_
id, total_amount, created_at, updated_at)
VALUES (PAGO_SEQ.NEXTVAL, :order_id, :payment_method_id,
0, :total_amount, SYSDATE, SYSDATE)
```

  - **Propósito:** Crear un registro de pago inicial para la orden.
- **create_shipping(cursor, order_id, shipping_address)**
  - **Consulta:**

```
INSERT INTO Envio (id, order_id, company_id, shipping_status_i
d, address, number_company_guide, dispatch_date, created_at,
updated_at)
```

```
VALUES (ENVIO_SEQ.NEXTVAL, :order_id, 1, 1, :address, 'G'||:ord
er_id, SYSDATE, SYSDATE, SYSDATE)
```

    - **Propósito:** Registrar un envío asociado a la orden, asignando un número de guía basado en el ID de la orden.

- **get_orders_from_db()**
    - **Consulta:**

```
SELECT
    o.id AS order_id,
    o.client_id,
    NVL(SUM(d.subtotal), 0) AS total_amount,
    o.created_at
FROM ORDENCOMPRA o
LEFT JOIN DETALLEORDEN d ON d.order_id = o.id
GROUP BY o.id, o.client_id, o.created_at
ORDER BY o.created_at DESC
```

    - **Propósito:** Listar todas las órdenes de compra junto con su monto total y fecha de creación.

- **get_order_by_id(order_id)**
    - **Consultas incluidas:**
        - **Datos generales de la orden:**

```
SELECT
    o.id,
    o.client_id,
    NVL(SUM(d.subtotal), 0) AS total_amount,
    o.created_at
FROM ORDENCOMPRA o
LEFT JOIN DETALLEORDEN d ON d.order_id = o.id
WHERE o.id = :order_id
GROUP BY o.id, o.client_id, o.created_
```

        - **Detalles de la orden:**

```
SELECT
    product_id,
    quantity,
    unit_price,
    subtotal
FROM DETALLEORDEN
WHERE order_id = :order_
```

- **Propósito:** Recuperar la información completa de una orden específica, incluyendo sus detalles (productos, cantidades, precios).

---

4. Módulo de Usuarios (`db_users.py`)

- **is_username_taken(conn, username)**
  - **Consulta:**

    ```
    SELECT COUNT(*) FROM CLIENTE WHERE username = :userna
    me
    ```

  - **Propósito:** Comprobar si el nombre de usuario ya existe.
- **is_email_taken(conn, email)**
  - **Consulta:**

    ```
    SELECT COUNT(*) FROM EMAILCLIENTE WHERE email = :email
    ```

  - **Propósito:** Verificar si el email ya está registrado.
- **insert_cliente(conn, ...)**
  - **Consulta:**

    ```
    INSERT INTO CLIENTE (
        id, national_document, name, lastname, username, phone, acti
    ve, password, created_at, updated_at
    )
    VALUES (
        CLIENTE_SEQ.NEXTVAL, :national_document, :name, :lastnam
    ```

```
e, :username, :phone, :active, :password, :created_at, :created_a
t
)
```

- **Propósito:** Registrar un nuevo cliente en la base de datos.

- **insert_email_cliente(conn, client_id, email, created_at, confirmed=1)**
  - **Consulta:**

```
INSERT INTO EMAILCLIENTE (
    id, client_id, email, confirmed, created_at, updated_at
)
VALUES (
    EMAILCLIENTE_SEQ.NEXTVAL, :client_id, :email, :confirmed, :
created_at, :created_at
)
```

  - **Propósito:** Asociar un email a un cliente.

- **get_client_by_username(conn, username)**
  - **Consulta:**

```
SELECT id, password FROM CLIENTE WHERE username = :user
name AND active = 1
```

  - **Propósito:** Obtener el ID y contraseña de un cliente a partir de su nombre de usuario.

- **get_client_by_id(conn, client_id)**
  - **Consulta:**

```
SELECT id, username, phone, created_at FROM CLIENTE WHER
E id = :id AND active = 1
```

  - **Propósito:** Recuperar la información de un cliente a partir de su ID.

- **update_client(conn, client_id, phone, email)**
  - **Consulta:**
    La sentencia SQL se construye dinámicamente según los campos a actualizar (teléfono y/o email).
  - **Propósito:** Actualizar la información de contacto de un cliente.
- **delete_client(conn, client_id)**
  - **Consulta:**

```
UPDATE CLIENTE SET active = 0 WHERE id = :id
```

  - **Propósito:** Desactivar un cliente marcándolo como inactivo.
- **insert_address(conn, client_id, address, created_at)**
  - **Consulta:**

```
INSERT INTO DIRECCION (
    id, client_id, address, created_at, updated_at
)
VALUES (
    DIRECCION_SEQ.NEXTVAL, :client_id, :address, :created_at, :c
reated_at
)
```

  - **Propósito:** Registrar una nueva dirección para un cliente.
- **get_payment_methods_by_client(conn, client_id)**
  - **Consulta:**

```
SELECT mp.id, mp.name
FROM MetodoPago mp
JOIN MetodoPagoCliente mpc ON mp.payment_client_id = mpc.i
d
WHERE mpc.client_id = :client_id
```

    - **Propósito:** Listar los métodos de pago disponibles para un cliente.
- **list_clients_with_payment_methods(conn)**
    - **Consulta:**
      Combina la consulta de clientes activos y la obtención de métodos de pago asociados mediante llamadas internas a `get_payment_methods_by_client`.
    - **Propósito:** Recuperar un listado de clientes junto con sus métodos de pago.